ARI REsearch Note 87-46

Operation of the World Master in the World Modeling System

Keith Barnett
Carnegie Mellon-University

for

Contracting Officer's Representative
Judith Orasanu

BASIC RESEARCH LABORATORY
Michael Kaplan, Director

U. S. Army

Research Institute for the Behavioral and Social Sciences

October 1987

U. S. ARMY RESEARCH INSTITUTE

FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency under the Jurisdiction of the

Deputy Chief of Staff for Personnel

EDGAR M. JOHNSON
Technical Director

WM. DARRYL HENDERSON
COL, IN
Commanding

Research accomplished under contract
for the Department of the Army

Carnegie-Mellon University

Technical review by

Dan Ragland

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ARI Research Note 87-46	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Operation of the World Master in the World Modeling System		5. TYPE OF REPORT & PERIOD COVERED Interim Report January 86 - January 87
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Keith Barnett		8. CONTRACT OR GRANT NUMBER(s) MDA903-85-C-0324
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department Carnegie-Mellon University Pittsburgh, PA 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 2Q161102B74F
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Research Institute for the Behavioral and Scoial Sciences, 5001 Eisenhower Avenue, Alexandria, VA 22333-5600		12. REPORT DATE October 1987 13. NUMBER OF PAGES 14
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* - -		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE n/a

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

- -

18. SUPPLEMENTARY NOTES

Judith Orasanu, contracting officer's representative

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Sensory-Effector Interface (SEI); Problem Solving;
Artificial Intelligence; World Modeling;
Machine Learning; Planning; simulators; mathematical models

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This research note describes the operation of the World Master process of the World Modeling System. It is intended to be an aid to maintainers of the World Master, and implementors of additional simulated physical properties of the world.

contd pg 1

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

Table of Contents

1. Introduction

The World Master is the core of the World Modeling System. It has two major functions: simulating the physics of a world, and keeping track of the status of all processes using a given world.

A world operates synchronously, with two major stages in a cycle of world time. During the first stage, the master must accept messages from other processes, and verify that they are in a format consistent with the world protocol (see *The World-Wide Communication Protocol of the World Modeling System*). Some messages will be requests to update the status of processes, and some will be requests to change data structures used in the simulation of the world. In the latter case, the master must not only note the changes made by storing them in a buffer, but send them immediately to all user interface (UI) processes, so that they may keep a consistent copy of the state of the simulation. The buffer will be sent later to all Sensory-Effector Interface (SEI) processes, which during this stage may be busy serving organisms.

The second stage of a cycle occurs after all processes using the world have indicated that they have no further requests to make in that cycle. During this stage, the master must examine the objects in the world, and update their state according to the physics of the simulation. All such changes must be recorded in a buffer; at the completion of this stage, the buffer will be sent to all UIs, and, along with the buffer created during the first stage, to all SEIs.

2. Message Handling

2.1 Adding and Removing Processes

The procedures for adding and removing processes from a world are outlined in *The World-Wide Communication Protocol.* In addition, the master must check and remove a process which disappears without properly exitting the system. A master attempts to remain connected to the World Coordinator at all times, but in general communicates with it only when a UI or the master itself is started.

A limited amount of information is kept for each process: the port to which messages are sent, or from which messages are received; the type of process (UI or SEI); the number of cycles it is prepared to accept without interrupting the master with requests; the process, if present, which was responsible for adding it to the system; and, for an SEI, information concerning the organism it serves, plus a flag indicating whether the SEI was added to the world in the current cycle. The master sends a UI an entire copy of the world immediately upon adding it; the UI will thereafter function just as any other. An SEI, however, is not fully active until the end of the cycle in which it is added, at which time it will be sent a copy of the world.

When an SEI is added, the master requests that a UI (the one requesting the addition) arrange that a graphics process be connected to the SEI, if appropriate, in order to simulate vision for the organism.

Related Protocol Commands: MasterStarted, AreYouThere, IAmAlive, WMExit, StartUI, AttachUI, StartUIReply, UIStarted, UIExit, ShutDownWorld, StartOrganism, AttachOrganism, StartOrgansimReply, OrganismStarted, OrganismExit, IAmYourMaster, SetOrgansimBody, SetOrganismName, TerminateOrganism, RequestToStartOrganism, RequestForOrganismVision

2.2 Rights

Many requests are critical in the sense that the requesting process expects a reply, and must be able to associate the reply with the request. A user making a change to a world data structure might never see the change, if a second process made a similar request at about the same time. Some requests may expect a response with no intervening messages. To allow for this, the master permits (and, in many cases, requires) processes to request send rights before making a request. While one process holds these rights, any other process making a request requiring rights will be placed on a FIFO queue; rights should therefore be held as briefly as possible. After a process has been granted rights, it is guaranteed to receive no messages from the master except those relating to the requests it sends.

Note that the master must continue processing messages from all processes at all times during the first stage of a cycle; processes are simply prohibited from sending certain types of requests unless they hold rights. For this reason, while a message may contain many requests (commands), the text of every request must be contained entirely within a single message.

All requests resulting in changes to the world data structures require rights. Each time a process release rights, the master sends all changes buffered while that process held rights to all processes using the world.

If the master detects a protocol error in a request of a process holding rights, the process will lose its rights.

Related Protocol Commands: RequestRights, GrantRights, ReleaseRights, Acknowledge

2.3 World States

If requested to do so, the master will save the state of the world at the end of the first stage of a cycle; it will do this automatically every few cycles as a checkpoint. The state of the world includes all world data structures (*this is not the same as object files maintained* by a UI) plus information concerning each organism using the world. This information includes a name, the object in the world corresponding to the organism, and a method for restarting the organism in its current state.

A previously saved state of the world may be restored at any time. The current state is cleared before loading an old state. An SEI in the system, however, having the same name as an SEI in the state file is polled to see if it can restore itself. All other SEIs are removed from the system; the unrestorable ones listed in the state file are restarted. UIs are unaffected.

States are referred to by state (not file) name; two special states are "creation", which simply clears the world, and a null state, which is the most recently saved state. See *Concepts and Nomenclature in the World Modeling System.*

Related Protocol Commands: SaveState, RestoreState, ClearWorld, RequestToStartOrganism, SetCheckpointPath, SetCheckpointFreq, SetCheckpointSaving, RestoreCompleted, UnableToRestore

2.4 Changes to World Data Structures

A process holding rights may request that changes be made to world data structures (see *The World-Wide Communication Protocol* and *The World Data Structure*). This is how an SEI effects the world for an organsim. Some object fields only the master may change: transforms, enclosures, and, for complex objects, mass, mass center, and buoyancy.

Related Protocol Commands: SetWorldName, SetDebug, SetTimeStep, SetWorldTime, SetContacts, SetAtmosphericViscosity, SetDirectedLight, SetAmbientLight, SetAmbientTemperature, CreateObject, SetObject, InsertNextObject, RemoveNextObject, RemoveSubObject, DeleteObject, CreateEmission, InsertNextEmission, RemoveNextEmission, DeleteEmission

2.5 Miscellaneous

Every process in the system must indicate that they are ready to proceed before the master will begin the second stage of a cycle. This is cumulative, and may be done for any number of cycles in advance.

At the end of the second stage of a cycle, after sending all buffered changes, the master must notify all processes of the start of the next cycle.

Any process may broadcast a message to all other processes in the world.

When the master detects an error in a request, it reports as much to the offending process, and, if the process held rights, broadcasts a message in order to alert other processes that changes about to be sent may be incomplete.

Upon request, the master will provide information about the processes it believes to be using its world.

Related Protocol Commands: OkToProceed, NotOkToProceed, StartNewCycle, DisplayMessage, ProtocolError, RequestProcessInfo

2.6 Protocol Modification in the World Master

The routine **HandleRequests()** in **handlereq.c** is the driver for the first stage of a cycle in the master. From there, routines are called in the same module to handle processing of messages, including most protocol commands. **SetUpProtocol()** in **handlereq.c** and **SetUpOProtocol()** in **communicate.c** must be modified to indiciate when new protocol commands are valid.

Routines to process changes to the world data structures and routines to handle the low-level mechanics of forming and sending messages are located in **communicate.c**. These are called from this module, from many places in **handlereq.c**, and from the driver **UpdateWorld()** in **updateworld.c**, where changes due to simulated physics are made.

See routine descriptions and comments in the code itself for more information.

3. Update of the World Data Structure

See *The World Data Structure* for a description of each type of structure and the fields represented.

3.1 Objects

The (translational and angular) acceleration, location, and velocity of each top-level (only) object are updated by the master at the beginning of the second stage of each cycle. The modification is done in that order: acceleration first so that forces applied during the first stage will have effect (acceleration is recomputed each cycle) and velocity last so that acceleration does not effect location twice in one cycle.

The acceleration of an object is the applied force divided by its mass, plus the acceleration due to gravity as modified by buoyancy.

All changes must be buffered to be sent to processes using the world.

In order to simulate lifting (which is difficult to do without support of collisions involving more than two *primitive* objects) the master allows (temporarily, one hopes) what is inappropriately called magnetic force. This is simply a force given to a primitive object; if the object comes into contact with a second object, the weight of which is less than the magnetic force, normal collision resolution is ignored, and the second object is attached to the top-level object to which the first object belongs (which can be the object itself). This attachment (done only in the master) results in a new rigid object, and is possible to break only by reducing the magnetic force.

3.2 Emissions

The master ignores emissions, except to remove them from the emission list when the duration of the emission has expired.

3.3 Modifying the World Update

World data structures are updated in the driving routine of the second stage, **UpdateWorld()** of **updateworld.c**. See the code for more information.

4. Collision Detection

4.1 Summary

The WM includes routines to detect collisions between polygons (polygonal surfaces), circles (discs), cylinders (right circular shells), spheres (spherical shells), and rigid objects consisting of combinations of these. There is currently no capability for jointed objects, and collisions involving more than two objects will not in general be resolved correctly. Collision detection algorithms decide on a single point of contact between objects; this point is an approximation due to the discrete time step used in the simulation, and due to the difficulty of resolving collisions along lines, planes, or multiple points of contact.

Objects are represented in the simulation by a hierarchical tree, with complex objects above their constituent parts, and primitive objects at the leaves. The representation of each object, complex or primitive, includes a matrix transform encoding the location, orientation, and scale of dimensions relative to the parent object (or to the global reference frame, in the case of top-level objects). This matrix should be invariant for all but the top-level objects, since all objects are rigid. There are also vectors representing the linear and angular velocity and acceleration -- zero for all but top-level objects -- and a field used by effecting organisms to indicate a force or torque to be applied at the center of mass of an object, and its duration. When joints are represented, effecting forces should be applied at the joints.

Collisions are represented for one time step for each object by a list of points of contact and force of contact with each (primitive) object involved. A collision between two top-level objects will be encoded as a single point of contact, whatever the actual geometry; which pair of a set of primitive objects in contact will be chosen to represent the overall contact will depend on the order in which objects appear in the tree structure used to represent objects.

Every object includes the radius of an enclosing sphere (used for efficiency in detecting collisions), and has a buoyancy, mass, and mass center of mass relative to its location. Primitive objects have coefficients of friction (kinetic only), restitution, and elasticity, and appropriate dimensions and (invariant) normals to surfaces. Polygons have a list of vertices represented as vectors relative to location.

4.2 Procedure

Detection algorithms are organized so that minimum effort is expended in cases where there is no contact between two objects, or where contact is easily determined. The radius of enclosure associated with each object allows pairs of objects whose radii do not overlap to be ignored.

Ideally, objects will collide in one point; because time is discrete, contact will ususally occur as overlapping volumes. Time should be interpolated to determine the first point of contact; this is not done. Depending on the type of overlap observed, assumptions are made as to the manner of initial contact, as described below.

4.2.1 Plane-Plane (Polygon and Circle)

The first test rules out pairs where the radius of enclosure of one object does not intersect the plane of the other. Otherwise, the perimeter of each object will intersect the line of intersection of the planes in an even number of points (single points taken to be two coincidental points). When ordered, these points mark the limits of areas of the objects; if any of these areas overlap, the objects are in collision.

The center of the longest such overlap is taken to be the point of contact. If the overlap corresponds exactly to an intersecting area of object A, it is assumed that object A "hit" object B, and the plane of contact is the plane of object B. Otherwise, it is assumed that edges of the two objects collided, and the plane of contact is tangent to both colliding edges.

4.2.2 Sphere-Sphere

If the radii overlap, the point of contact is taken to be midway between the overlap, on the line between the centers; the plane of contact is perpendicular to this line.

4.2.3 Plane-Sphere

If the sphere intersects the plane, it does so in a circle. The point of the plane object nearest the center of this circle (if the point is not in the circle, there is no collision) is taken as the point of contact, unless the plane is a circle with overlapping radius (see *Sphere-Sphere*). The plane of contact is tangent to the sphere at the point of contact.

4.2.4 Polygon-Cylinder

There is no collision if the cylinder enclosure does not intersect the plane of the polygon, or if the polygon enclosure if farther from the cylinder axis than the radius of the cylinder. Otherwise, that portion of the polygon within the height of the cylinder may be projected onto a plane perpendicular to the axis of the cylinder; the circular intersection of the cylinder may be treated as the *Polygon-Sphere* case to detect collision.

If the center of the circular intersection is not in the interior of the polygonal projection, an edge of the polygon is assumed to have "hit" the side of the cylinder, and the plane of contact is tangent to the cylinder at the point of contact (which will be on an edge of the polygon). Otherwise, an edge of the cylinder is assumed to have "hit" the polygon; the point of contact is the point of the polygon projecting to the center of the cylinder, and the plane of contact is the plane of the polygon.

4.2.5 Sphere-Cylinder

There is no collision if the sum of the radii is greater than the distance between the sphere center and the cylinder axis, if the sphere is farther "above" the height of the cylinder than the radius of the sphere, or if one contains the other (if the line segment along the side of the cylinder parallel to its axis and farthest from the center of the sphere is inside the sphere, but does not intersect the shell of the sphere, the cylinder is contained by the sphere).

In the last case, the point of contact is the point of intersection of the line segment with the sphere (nearest the cylinder center if there are two). In all other cases, collision reduces to that of *Sphere-Sphere*, using circles of intersection in a plane perpendicular to the cylinder axis, through an end of the cylinder or the sphere center, whichever is nearer the cylinder center. The plane of contact is tangent to the sphere at the point of contact.

4.2.6 Cylinder/Circle-Cylinder

This is messy. There is no collision if the sum of the radii is greater than the perpendicular distance between the axes, or if one object is a circle and the cylinder enclosure does not intersect its plane. Otherwise, the line passing through both axes and perpendicular to both is computed, and its points of intersection with the cylinders found. If the line does not intersect a cylinder, or the pairs of intersecting points do not overlap, there is no collision.

If the pairs of points are interleaved, the point of contact is taken to be the center of the overlap, and the plane of contact is perpendicular to the above line (tangent to the cylinders). Otherwise, one cylinder might be contained within the other, which is where it gets messy: collision reduces to that of

a circle with an ellipse. This can be solved with a quartic equation; unfortunately, I could not find robust solutions to quartic equations. I won't attempt to describe here the algorithm detecting containment.

4.3 Modifying Collision Detection

Collision detection, called from **UpdateWorld()**, is carried out by routines in **updateworld.c**. See routine definitions and comments in the code for more information. Routines to detect collisions between cylinders, which should probably be rewritten, can be found in **cylinders.c**.

5. Collision Resolution

5.1 Summary

Given that the following assumptions and approximations are made, collision resolution is reasonably accurate, accounting for three-dimensional resolution of linear and angular forces.

- All collisions occur between two isolated primitive objects. No attempt is made to model multiple collisions, including instances of stacked objects at rest. The results of such collisions will depend on the order in which they are resolved.
- The point and plane of contact are approximations at best and arbitrary at worst.
- The coarser the time scale, the less accurate detection and resolution will be.
- A primitive object has uniform density.
- Associated with each primitive object are a modulus of elasticity and a coefficient of restitution, as measured in collisions with an object of the same material.
- Associated with each primitive object is a single, constant coefficient of friction; the friction used in a collision is the average of the frictions of the two objects involved.
- The inertia of a polygonal surface is approximated by that of a disc with radius 20% smaller than the enclosure of the polygon.
- It is assumed that an immense, massive circle (or similar object) with zero buoyancy will serve as a base to the world.

The collision resolution algorithm is based on a discussion in *Impact*, by Werner Goldsmith, pp. 11-17. (Edward Arnold (Publishers), ltd., London 1960.)

5.2 Procedure

During every cycle, the accelerations of each object are set using effector requests and a gravitational constant. Displacements are modified, using the new accelerations and the velocities calculated during the previous cycle. Velocities are modified using the new accelerations. Finally, collision resolution takes place. This order is important. For instance, if velocities were updated before displacements, a falling object which hit the floor in the previous cycle might have its resulting upward velocity negated by gravity during the current cycle, and fall through the floor. If collision detection were done first, an organism would not be able to react to collisions without the delay of a cycle.

Collisions are resolved by calculating the impulse generated, and using this to compute final velocities.

An impulse may be split into components parallel and perpendicular to the plane of contact; the parallel impulse is the frictional force. For smaller coefficients of friction or lesser perpendicular impulses (to which the frictional force is proportional) sliding will occur in the plane of contact. In other cases, the frictional force may become great enough to eliminate (perhaps temporarily, if the collision is elastic) relative velocities between the colliding objects, in the plane of contact.

The magnitudes of the initial relative velocities in the direction of compression determine the magnitude of the greatest possible perpendicular impulse, defining a plane of greatest compression. Similarly, the magnitudes of the initial relative velocities in the plane of contact define a line of no sliding, where relative velocity becomes zero. The path followed in impulse space as a function of time lies along a cone defined by the coefficient of friction; the points at which this path intersect the plane of compression and line of no sliding determine the value of the final impulse. It seemed a bit much to try to solve the set of third order differential equations defining this path, so it is approximated by a straight line.

The impulse is computed by determining where the friction cone intersects the line of no sliding and plane of greatest compression. For high coefficients of friction, the impulse path will reach the former first, and continue along the line (the friction then being great enough to prevent sliding). Otherwise, when the latter is reached, the path will appear to be "reflected", as the compression force begins to decrease. The length of the path following intersection with the plane of greatest compression depends on the elasticity of the collision.

5.3 Modifying Collision Resolution

Good luck. Called from **UpdateWorld()** with a point and plane of contact, resolution is handled by **ResolveCollision()** and other routines in **resolve.c**. See the code for more information.